ATYPON

Instructors:
Mutasim Aldiab
Fahed Jubair

done by:

Hamza Hassan

Student Grading System Assignment

ABSTRACT

In this report, we will talk about building a student grading system
This will be done in three ways divided into three parts
He will focus on backend development, web application development, and Spring MVC/Spring REST
In the first part, we will build client-server architecture.
In the second part, we will build a Web app using Servlet.
In the third part, we will build it with a spring boot.

Introduction

In this report, we will talk about the difference in dealing with Java.
The primary objective of this assignment is to develop a robust and efficient Student Grading System that encompasses different technological aspects of backend development and web application architecture
The content will cover the necessary steps for completing this assignment, including discussing the required tasks and outlining the construction of technological components, while also highlighting the distinctions between them.

Database

Let's delve into the topic of student grading systems in a broader context and discuss the database components. The system comprises five distinct tables within the database:

1. Students Table:
   This table stores information about individual students

student_id	username	password	full_name
1	hamza11	pass11	hamza hassan
2	ali11	pass11	ali Student
NULL	NULL	NULL	NULL

2. Instructors Table:
   This table stores information about individual instructor

instructor_id	username	password	full_name
1	instructor1	pass11	Fahed Instructor
2	instructor2	pass22	Mutasim Instructor
NULL	NULL	NULL	NULL

3. Courses Table:
   This table stores information about courses

course_id	course_name	instructor_id
1	object oriented programming	1
2	devops	2
3	c++	2
4	math	1
5	python	2
NULL	NULL	NULL

4. Grades Table:

grade_id	student_id	course_id	grade
1	1	1	79.1
2	2	1	80.6
3	2	2	55.5
4	1	2	80.3
5	1	3	44.2
6	2	3	99
13	1	4	60.1
17	1	5	88.3
18	2	4	60
NULL	NULL	NULL	NULL

5. Admin

admin_id	username	password	full_name
1	admin	admin	hamza admin
NULL	NULL	NULL	NULL

   These tables collectively manage student and instructor information, course details, grade records, and administrative access, facilitating efficient tracking and management of academic data.

After see what database we have now let's talk about part 1 (server-client architecture) in details

System functionality

For all parts in this assignment the functionality of the system is the same.

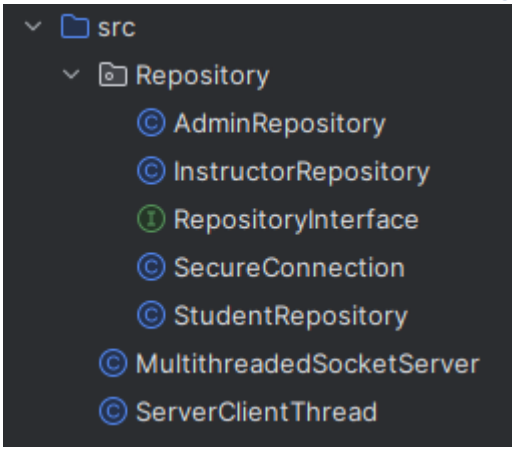
This system was created to serve users (students, instructor, admin) and make it easy for them.
It gives **students** the possibility for :
1. see his marks in the courses he is registered with.
2. display of statistical data (e.g., class average, median, highest, and lowest marks).

It gives **instructors** the possibility for:
1. The instructor can see the courses he teaches, and the students enrolled in it and their grades.
2. puts grade to the students in his courses.

It gives **admin** the possibility for:
1. Add Student to system
2. Delete Student from system
3. Add Instructor
4. Delete Instructor
5. Add Course
6. Delete Course
7. Register Student for Course
8. Appoint Instructor for Course
9. View Grades

Part 1: Command-line / Sockets and JDBC Backend Implementation

first let's take a look at OOP design used in this part :



1) Implemented the DAO design pattern by crafting a **RepositoryInterface** interface, followed by the creation of three distinct classes for streamlined database data retrieval:

1. **StudentRepository**
2. **InstructorRepository**
3. **AdminRepository**

This design approach enhances data organization and retrieval, aligning with the tenets of the DAO pattern.

2) MultithreadedSocketServer class, functions as a server that facilitates communication with clients using sockets. It enables multiple clients to connect

concurrently and interact with a MySQL database using secure credentials.

The server initiates on a specific port (8888) using a TCP ServerSocket, which listens for incoming client connection requests.

Once a client request is received, the server accepts the connection and spawns a new thread (**ServerClientThread**) to handle the client's communication.

3)ServerClientThread class: used to handle interactions between clients and a server for a Student Grading System
This class extends the Thread class, which means it can be used to create multiple threads that handle client-server communication simultaneously.
Student Interaction: If the client is identified as a student (input is "1")
Instructor Interaction: If the client is identified as an instructor (input is "2")
Admin Interaction: If the client is identified as an admin (input is "3")

4)ScureConnections class : is responsible for establishing a secure connection to a MySQL database. This review aims to analyze the class's structure, its security aspects. its use JDBC driver to connection with database, It is not possible to access the username, password, and the URL of Database, except through using getters.

5)client side : Proper command-line implementation and handling of user input. Whether he is a student, instructor , or admin, and give him the tasks that he can perform and leave the choice to him

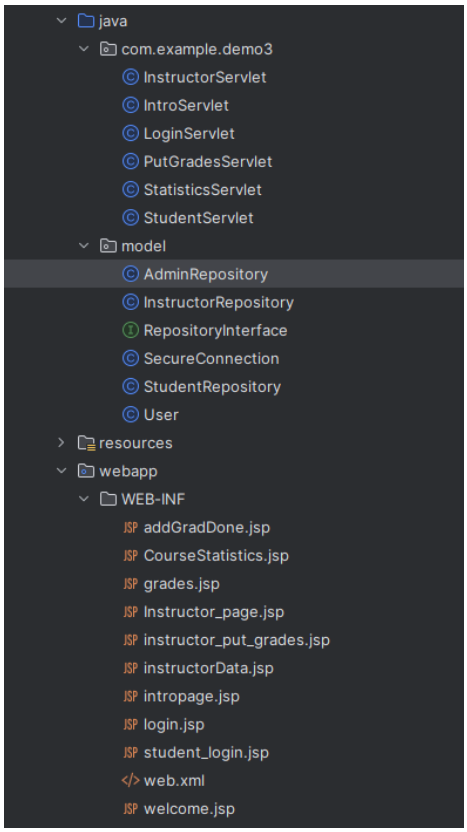
Sockets and JDBC communication between the client and server is handled using TCP sockets. The client initiates a connection to the server, and the server responds accordingly. JDBC is used to interact with the database for data storage and retrieval.
This approach requires manual handling of socket connections, threading, and managing low-level communication details. JDBC involves writing SQL queries, managing database connections, and handling result sets.
Code can become complex and hard to maintain due to manual handling of connections, threading, and SQL queries.
Scaling can be complex due to manual handling of threads and connections. Load balancing and managing concurrent connections require careful planning.

SO now we will move to approach will handling the requests, less than complex called Servlets

Part 2: Web App / Traditional MVC Servlets and JSPs Implementation

This approach involves writing Servlets to handle requests, manage business logic, and render JSPs for views.

first let's take a look at OOP design used in this part :



To implement a web application based on traditional MVC design pattern, we'll create the *Repository* classes which will act as our **Model** layer.
Servlets classes will act as a **Controller**, and for the presentation layer, we'll create *jsp* page.

How it works?

First, when the user interact with JSP page the servlet reads a parameter from the request. If the request submitted, then is fetched from the MODEL layer.

Once it retrieves the necessary data from the Model, it puts this data in the request using the *setAttribute()* method.
Finally, the Controller *forwards* the request and response objects to a JSP, the view of the application.

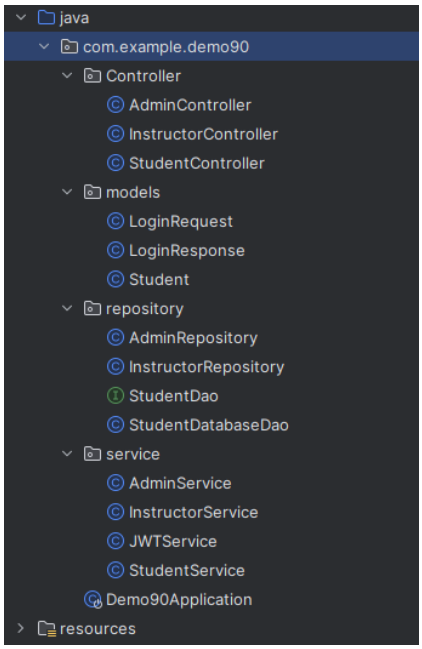
1) Implemented the DAO design pattern by crafting a **RepositoryInterface** interface,like what we do in the first part with the same implementation to deal with data from database
2) Every model has the **Servlets** classes to handles incoming requests from clients (usually web browsers), processes these requests, and generates appropriate responses. It serves as the middleman between the user interface and the data model, ensuring that the correct actions are taken based on user interactions. Then view the response using JSP page
3) JSP To help deal with the servlet by taking requests and re-showing the response to the client in web page ,

Part 3: Web App / Spring MVC and Spring REST Implementation

design:

For Decoupling and Cohesion and Follow the design principle "Separation of Concerns."
I split the code like this
create four package :

- ✓ Controller : to put all the spring boot classes on it
  - Need to create controller to all potential users
- ✓ Models : need one model to login request and one to the login response to can were the response will contain a token to secure the login information
- ✓ Service : needed to create some classes that provide the required services to the controller after taken them from the Repository
- ✓ Repository



For secure user authentication and authorization, generating JSON Web Tokens (JWTs), and for authenticated. Let's discuss the **security aspects** :

1. Authentication and Authorization:
The code implements a basic form of authentication by allowing user to log in using a POST request to /students/login. It compares the provided username and password against some authentication logic within studentService.authenticate(). If successful, it generates a JWT using jwtService.generateToken() and returns it in the response. The JWT will include the username of the authenticated student.

2. JWT Generation:
JSON Web Tokens (JWTs) are being used for authentication. JWTs are a secure way to transmit information between parties as a JSON object. They consist of three parts: header, payload, and signature. The JWT is signed using a secret key. In this code, the jwtService is responsible for generating and validating JWTs.

3. Token Usage:
The JWT is included in the header of subsequent requests, specifically in the Authorization header. The token is prefixed with hamza and is extracted and processed in the welcomeMessage() and displayGrades() methods.

what the difference between 3 part ?

Sockets and JDBC backend compared to Traditional MVC (Servlets and JSPs) and Spring MVC / Spring REST
Its drawbacks appear in:

- Sockets require manual management of network communication, leading to complex code.
- Handling security features like authentication and encryption can be challenging with sockets.
- JDBC involves writing repetitive code for database connections and queries.

In this third part we replaced the **servlet** with the **spring REST** implementation and we will replace **traditional MVC** with **spring MVC.**

what is the difference ??

spring MVC it is very easy to enable a rest back-end for our web application. It's a web framework provided by the Spring Framework that facilitates the development of web applications. It follows the MVC architectural pattern, as we mentioned in part 2
the difference between them is
1. Spring MVC leverages Spring's dependency injection, making it easier to manage dependencies and write testable code.
2. Spring MVC offers more flexibility in terms of choosing view technologies, integrating with various data sources, and handling different types of requests.
3. Spring MVC promotes standardized URL mapping and request handling, making it easier to create RESTful APIs and manage routes.

the difference between servlet and spring boot:

Servlets:

1. Requires manual URL mapping and routing setup.
2. Manual parsing and processing of request data.
3. Offers flexibility but requires more coding for handling different types of requests and data formats.
4. Developers need to manage response headers, status codes, and streams.
5. Configuration is typically done using the web.xml file or annotations.
6. Does not provide built-in integration with other frameworks.

Spring REST :

1. Automatic routing based on annotations, simplifying URL mapping.
2. Automatic data serialization and deserialization for various formats.
3. Offers flexibility while providing abstractions for common tasks.
4. Provides simple ways to set response codes, headers, and more.
5. Configuration through annotations and configuration files like application.properties.
6. Seamlessly integrates with the Spring ecosystem and other Spring features.

Lets take a look in my code to see what I talking about
sprig boot controller (for example StudentController Class) and
the servlet controller (for example StudentServlet Class)

Servlet

```java
@WebServlet(urlPatterns ="/marks")
public class StudentServlet extends HttpServlet {
    @Override
    public void init(ServletConfig config) throws ServletException {
        SecureConnection secureConnection=new SecureConnection();
secureConnection.getSecureConnection();
        super.init(config);

    }

    no usages
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        HttpSession session = req.getSession();
        User loggedInUser = (User) session.getAttribute( s: "loggedInUser");
        StudentRepository studentRepository = null;
        try {
            studentRepository = new StudentRepository(loggedInUser.getUsername(), loggedInUser.getPassword());
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }

        boolean exit=false;
        while (!exit){
        String choice= req.getParameter( s: "choice");
            switch (choice) {
                case "1":
                    System.out.println("choice 1");
                    String marks = studentRepository.display();

                    req.setAttribute( s: "marks", marks);
                    // req.setAttribute("id",id);
                    System.out.println(marks);
                    req.getRequestDispatcher( s: "WEB-INF/welcome.jsp").forward(req, resp);
                    System.out.println("end choice 1");

                    break;
                case "2": {
```

Spring REST

```java
@RestController
@RequestMapping("/students")
public class StudentController {

    4 usages
    private final StudentService studentService;
    4 usages
    private final JWTService jwtService;

    @Autowired
    public StudentController(StudentService studentService, JWTService jwtService) {
        this.studentService = studentService;
        this.jwtService = jwtService;
    }
    @PostMapping("/login")
    public ResponseEntity<LoginResponse> login(@RequestBody LoginRequest loginRequest) {
        if (studentService.authenticate(loginRequest.getUsername(), loginRequest.getPassword())) {
            String token = jwtService.generateToken(loginRequest.getUsername());
            return ResponseEntity.ok(new LoginResponse( message: "Login successful!", token));
        } else {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(new LoginResponse("Login failed!"));
        }
    }

    @GetMapping("/welcomeMessage")
    public String welcomeMessage(@RequestHeader("Authorization") String tokenHeader) {
        String token = tokenHeader.replace( target: "hamza ",  replacement: "");
        String username = jwtService.getUsernameFromToken(token);
        return studentService.welcomeMessage(username);
    }

        @GetMapping("/displayGrades")
    public String displayGrades(@RequestHeader("Authorization") String tokenHeader) {
        String token = tokenHeader.replace( target: "hamza ",  replacement: "");
        String username = jwtService.getUsernameFromToken(token);

        return studentService.display(username);
    }
```

- Annotation-based Configuration:

Spring Boot Code: Relies on annotations for configuration, routing, dependency injection, etc.
Servlet Code: Uses manual servlet configuration via annotations and web.xml.

- Dependency Injection:

Spring Boot Code: Utilizes Spring's dependency injection for managing components and services.
Servlet Code: Manually initializes and manages objects.

- Routing:

Spring Boot Code: Leverages annotations like @GetMapping and @PostMapping for routing.
Servlet Code: Manually maps URLs to servlets via @WebServlet annotation.

- Database Interaction:

Spring REST Code: Uses JDBC template or JPA for database interactions.
Servlet Code: Uses manual JDBC driver or other database access methods.
that's move as from this in manual JDBC driver:
as we can see below use JDBC template is more easer. JDBC template is provided to us by spring boot

```
1 usage
public void addStudent(String fullName, String username, String password) {
    String sqlQuery = "INSERT INTO students (full_name, username, password) VALUES (?, ?, ?)";
    jdbcTemplate.update(sqlQuery, fullName, username, password);
}

1 usage
public void deleteStudent(int studentId) {
    String sqlQuery = "DELETE FROM students WHERE student_id = ?";
    jdbcTemplate.update(sqlQuery, studentId);
}
```

```
1 usage
public void addStudent(String fullName, String username, String password) {
    try {
        String sqlQuery = "INSERT INTO students (full_name, username, password) VALUES (?, ?, ?)";
        PreparedStatement preparedStatement = connection.prepareStatement(sqlQuery);
        preparedStatement.setString( parameterIndex: 1, fullName);
        preparedStatement.setString( parameterIndex: 2, username);
        preparedStatement.setString( parameterIndex: 3, password);
        preparedStatement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

// Delete a student from the database
1 usage
public void deleteStudent(int studentId) {
    try {
        String sqlQuery = "DELETE FROM students WHERE student_id = ?";
        PreparedStatement preparedStatement = connection.prepareStatement(sqlQuery);
        preparedStatement.setInt( parameterIndex: 1, studentId);
        preparedStatement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

The end